DEBUGGING FORTRAN MEMORY PROBLEMS IN AIX

George VandenBerghe
IBM NCEP SUPPORT
sp-support@ncep.noaa.gov

CONTENTS

- Define segmentation fault
- Describe memory allocation in fortran.
- Illustrate code forms that produce memory bugs.
- Describe helpful compiler and load options to find memory bugs.
- Describe some helpful runtime environment variables
- Describe some manual debugging techniques
- Introduce dbx and Totalview but NOT in detail.

What is a segfault?

Segfaults occur when an instuction uses an invalid address. They can occur in legal code when your data structures are too large.

Bug segfaults are usually due to memory overwrites which change address lengths in executable code, overwrite array index variables or array descriptors in subroutines.

Best way to prevent these is to find the causes (addressed in coming slides).

Segfaults from oversize data structures occur in failing routine. Segfaults from memory overwrites are often far from failing routine

MEMORY ALLOCATION

- Memory is allocated from either a full data area or a subset called the STACK.
- Default data area on asp/bsp is 128mb and default stack is 32mb.
- These can be increased at load time with load options – *bmaxdata:size –bmaxstack:size2.*
- Max is *–bmaxdata:0x80000000 –bmaxstack:0x10000000*

Decimal equivalents are 2gb and 256mb or - *bmaxdata:2000000000 –bmaxstack:256000000*

These are slight underestimates.. Hex values are actual maxes

Frost and snow values are huge for data and 0x80000000 for stack

ERRORS FROM INSUFFICIENT MEMORY

a.out

1*: exec(): 0509-036 Cannot load program ./a.out because of the following errors:*
0509-026 System error: There is not enough memory available now.

2: *1525-108 Error encountered while attempting to allocate a data object. The program will stop.* (this can have many other causes however)

```
c       dimension array(262000,300)   (case 1 error produced when allocatable
      real , allocatable :: array(:,:)      and allocate are commented out and
      allocate(array(262000,300))           dimension  comment is removed)
     array=5
          print *,'hello world'
      stop
      end
```

ERRORS FROM INSUFFICIENT MEMORY

Programs asking for too much memory may also segfault in Subroutine initialization sections, in MPI or (rarer) in I.O. Libraries.

Correct by loading with larger –bmaxstack: and –bmaxdata.

WHAT GOES WHERE

SUBROUTINE SUB(V,IX)
COMMON/BLK/C(100) <<*common DATA*
DIMENSION A(100) << *subroutine local varies with compiler and options*
DIMENSION B(IX) <<*automatic.. ALWAYS stack*
DIMENSION V(IX) <<*depends on caller allocation*
REAL, ALLOCATABLE :: V2(:)

ALLOCATE (V2(IX)) <<*allocatable always DATA*

COMPILER OPTIONS CONTROLLING LAYOUT

- -qsave makes subroutine local variables static (DATA)
- -qnosave makes these STACK
- Static is default for f77 xlf and mpxlf
- Stack is default for xlf_r xlf90 mpxlf_r mpxlf90 and mpxlf90_r.
- Avoid qsave for new programs. Trend is away from this as default. Use SAVE of variables that must retain values between calls. COMMON blocks created in subroutines must also be SAVED
- Variables initialized with a DATA statement are static. This behavior is vendor dependent.

MEMORY OVERWRITE BUGS

- These are common and HARD to find.
- Most common cause of segfaults
- Senior analysts will remember MVS error 240.. Segfaults are analagous.
- Two common forms

ARRAY OUT OF BOUNDS

```
dimension A(100)
Do k=1,200
A(k)=value
End do
```

Whatever variable is stored after A will get silently clobbered.

Compile with **–qcheck** or **–C** finds these at runtime

Executable aborts with trace/bpt trap if compiled with –qcheck.

SUBROUTINE BINDING MISMATCH

- Real a,b,c

call sub(a,b,c)

Subroutine sub(a,b,c)

Real(kind=8) c,a

Real b

Stores into a and c will clobber b and something after c.

- Invisible array length descriptors can get clobbered this way also.

-qextchk options

- Subroutine binding errors cannot generally be caught by compiler in fortran 77 code. *(modules and CONTAINED subroutines enable detection in fortran 90)*
- The linker can detect them with help
- -qextchk in compile AND link steps causes f77 code to be checked for binding errors at load time (inconsistent calls in same routine are caught by COMPILER a special fortran 77 case)

-qextchk

- -qextchk detects binding errors at link time.
- Loader terminates with return code 8
- If –bloadmap:file is specified then a text file containing all of the mismatched routine names and their callers will be written. (this file is used for other loader output also)
- Common block alignment errors are also sometimes caught.

-qextchk PROBLEM

- Binding mismatches are allowed in fortran.
- Routines can be called with parameters of different types in different calls.
- MPI routines are generally called with different types for some arguments. Legal and correct MPI code is unlikely to load properly with –qextchk.

One can go through loadmap and exclude MPI names from consideration or examine these for actual binding ERRORS rather than mismatches.

-qextchk

- An alternative is to insert
- @PROCESS NOEXTCHK before MPI calls.

RUNTIME OPTIONS

- For MPI jobs try buffersize variables.
- For threaded jobs check threadcount and thread buffer variables sizes. (XLSMPOPTS="parthds=4,stack=64000000")
- *(default stack setting of 4mb is common cause of segfaults in large threaded executables)*
- Codes that try to write huge corefiles may instead write useless zero length corefiles.
- If this happens try MP_COREFILE_FORMAT=lite in environment.
- Try setting MP_SHARED_MEMORY to no (default on asp/bsp is no but on frost/snow yes)

RUNTIME OPTIONS

- Try setting MP_EAGER_LIMIT to zero (this prevents buffering of MPI messages and associated buffer issues but performance for small messages is lowered. *May also cause deadlocks but these are user errors in comms design which would have surfaced anyway*)
- Try setting MP_BUFFER_MEM to lower or higher values (default is 2.8 mbytes). Lower values reduce performance but also reduce buffer memory problems.
- If segfaults produce zero length corefiles, try setting MP_COREFILE_FORMAT=lite. This will write a compact TEXT corefile with a trace.

MANUAL METHODS (faster than you think!)

- For optimization problems try compiling half of .f with high and half with no optimization. Redo switching halves.
- Repeat with half that fails. (this is called a binary search)
- This method often isolates single routines with optimization bugs. Convergence is logarithmic.
- Technique also works for segfaults when a good variable address goes bad. Prints of bad variable work until the clobbering occurs. It also works for memory overwrites that occur early in execution and also for isolating numerical differences

Debuggers

Dbx. Most useful for a quick look at a corefile.

Do cd to core containing directory and do

dbx a.out where a.out is your executable.

At prompt type "where" to get stack trace.

TOTALVIEW DEBUGGER IS BEYOND SCOPE OF THIS TALK (But TOTALVIEW is available on frost/snow!)

When Presenting Problems!

- Please send any problems to ***sp-support@ncep.noaa.gov*** Problems sent privately to someone on the NCEP team may not be tracked or resolved as optimally.